PDL for ordered trees

Loredana Afanasiev* — **Patrick Blackburn**** — **Ioanna Dimitriou***
Bertrand Gaiffe** — **Evan Goris*** — **Maarten Marx***
Maarten de Rijke*

* *Informatics Institute*
University of Amsterdam,
Kruislaan 403,
1098 SJ Amsterdam (The Netherlands)
{lafanasi,idimitri,egoris,marx,mdr}@science.uva.nl

** *INRIA Lorraine*
615, rue du Jardin Botanique,
54600 Villers les Nancy Cedex (France)
{patrick.blackburn,bertrand.gaiffe}@loria.fr

ABSTRACT. *This paper is about a special version of PDL, proposed by Marcus Kracht, for reasoning about sibling ordered trees. It has four basic programs corresponding to the child, parent, left- and right-sibling relations in such trees. The original motivation for this language is rooted in the field of model-theoretic syntax. Motivated by recent developments in the area of semi-structured data, and, especially, in the field of query languages for XML (eXtensible Markup Language) documents, we revisit the language. This renewed interest comes with a special focus on complexity and expressivity aspects of the language, aspects that have so far largely been ignored. We survey and derive complexity results, and spend most of the paper on the most important open question concerning the language: what is its expressive power? We approach this question from two angles: Which first-order properties can be expressed? And which second-order properties? While we are still some way from definitive answers to these questions, we discuss two first-order fragments of the PDL language for ordered trees, and show how the language can be used to express some typical (second-order) problems, like the boolean circuit and the frontier problem.*

KEYWORDS: *pdl, ordered trees, definability, expresivity, complexity.*

1. Introduction

The purpose of this paper is to revive interest in a version of PDL proposed by Marcus Kracht [KRA 95, KRA 97]. This version, called $\mathsf{PDL}_{\mathsf{tree}}$ here, is specially designed for models which are sibling ordered trees. Such models are of interest in at least two research communities: linguistics, in particular the field of model-theoretic syntax, and computer science, in particular for those working with the World Wide Web, semi-structured data and XML (eXtensible Markup Language).

Model-theoretic syntax is an uncompromisingly declarative approach to natural language syntax: grammatical theories are logical theories, and grammatical structures are their models. These models consists of parse trees, i.e., node labeled, sibling ordered finite trees. Perhaps the best known work in this tradition is that of James Rogers (for example [ROG 98]) in which grammatical theories are stated in monadic second-order logic. However other authors (in particular [BLA 94, KRA 95, KRA 97, PAL 99]) use various kinds of *modal logic* (in essence, variable free formalisms for describing relational structures) to specify grammatical constraints. Palm [PAL 99] contains some interesting linguistic examples and is a good introduction to (and motivation for) this approach.

The World Wide Web is a freely evolving, ever-changing collection of data with flexible structure. The Web's nature escapes the conventional database scenario of manipulating data: data on the Web simply do not comply with the strict schemas used for conventional databases. Web data such as home pages, news sites, pages on commercial sites, usually enjoy some amount of structure, but that is not strictly enforced, and there are no uniformly adopted standards, not even for simple bits of information such as addresses. Hence, data on the Web is essentially semi-structured [ABI 00]. In search for suitable models for semi-structured data, the World Wide Web Consortium proposed the eXtensible Markup Language (XML) [CON 04]. XML is a standard for textual representation of semi-structured data and was designed to describe any type of textual information. It looks like a flexible variant of HTML, allowing for the mark-up of data with information about its content rather than its presentation. The logical abstraction of an XML document (the so-called DOM) is a finite, node labeled, ordered tree.

Motivated by the renewed need for clean, well-understood declarative tree description formalisms brought about by the developments in semi-structured data outlined above, we want to revive interest in the special variant of PDL developed for sibling ordered trees. We focus on complexity and expressivity aspects of the language. Section 2 introduces the language. Section 3 discusses complexity, and in Section 4 and Section 5 we address expressivity issues. Section 4 is devoted to the expressiveness of the language in terms of first-order properties; we discuss the first-order fragment of $\mathsf{PDL}_{\mathsf{tree}}$, recall some known results, and show the language in action by expressing the until modality over the document order relation.

It follows from the failure of Beth's Theorem for deterministic PDL interpreted on finite trees [KRA 99] that $\mathsf{PDL}_{\mathsf{tree}}$ is strictly less expressive than unary monadic second-order logic (MSO). The most pressing issue thus is to determine the exact expressive power of $\mathsf{PDL}_{\mathsf{tree}}$ in terms of a suitable fragment of unary MSO. This remains an open problem, but to improve our understanding of $\mathsf{PDL}_{\mathsf{tree}}$'s expressive power, we adopt a well-known strategy by examining a number of 'typical' second-order problems and properties. Specifically, in Section 5 we show how we can express the boolean circuit and the frontier problem, and we discuss infinity axioms. These examples suggest that $\mathsf{PDL}_{\mathsf{tree}}$ is expressive enough to encode natural hard second-order problems. Boolean circuits is one of the main problems used to show that a logic is weaker than MSO. The frontier problem is a typical linguistic application. We conclude in Section 6.

2. PDL for ordered trees

We recall the definition of $\mathsf{PDL}_{\mathsf{tree}}$ from [KRA 95, KRA 97]. $\mathsf{PDL}_{\mathsf{tree}}$ is a propositional modal language identical to Propositional Dynamic Logic (PDL) [HAR 00] over four basic programs `left`, `right`, `up` and `down`, which explore the left-sister, right-sister, mother-of, and daughter-of relations. Recall that PDL has two sorts of expressions: programs and propositions. We suppose we have fixed a non-empty, finite or countably infinite, set of atomic symbols A whose elements are typically denoted by p. $\mathsf{PDL}_{\mathsf{tree}}$'s syntax is as follows, writing π for programs and ϕ for propositions:

$$\begin{array}{lcl} \pi & ::= & \texttt{left} \mid \texttt{right} \mid \texttt{up} \mid \texttt{down} \mid \pi;\pi \mid \pi\cup\pi \mid \pi^* \mid \phi? \\ \phi & ::= & p \mid \top \mid \neg\phi \mid \phi\wedge\phi \mid \langle\pi\rangle\phi. \end{array}$$

We sometimes write $\mathsf{PDL}_{\mathsf{tree}}(\mathrm{A})$ to emphasize the dependence on A. We employ the usual boolean abbreviations and use $[\pi]\phi$ for $\neg\langle\pi\rangle\neg\phi$.

We interpret $\mathsf{PDL}_{\mathsf{tree}}(\mathrm{A})$ on *finite ordered trees* whose nodes are *labeled* with symbols drawn from A. We assume that the reader is familiar with finite trees and such concepts as 'daughter-of', 'mother-of', 'sister-of', 'root-node', 'terminal-node,' and so on. If a node has no sister to its immediate right we call it a last node, and if it has no sister to its immediate left we call it a first node. The root node is both first and last, and called root. A labeling of a finite tree associates a subset of A with each tree node.

A sibling ordered tree is a structure isomorphic to $(N, R_{\mathtt{down}}, R_{\mathtt{right}})$ where N is a set of finite sequences of natural numbers closed under taking initial segments, and for any sequence s, if $s\cdot k \in N$, then either $k = 0$ or $s\cdot k-1 \in N$. For $n, n' \in N$, $nR_{\mathtt{down}}n'$ holds if, and only if, $n' = n\cdot k$ for k a natural number; $nR_{\mathtt{right}}an'$ holds if, and only if, $n = s\cdot k$ and $n' = s\cdot k+1$. We present finite ordered trees (trees for short) as tuples $\mathbf{T} = (T, R_{\mathtt{down}}, R_{\mathtt{right}})$. Here T is the set of tree nodes and $R_{\mathtt{right}}$ and $R_{\mathtt{down}}$ are the right-sister and daughter-of relations, respectively. A pair $\mathfrak{M} = (\mathbf{T}, V)$, where $\mathbf{T}$ is a finite tree and $V : A \longrightarrow Pow(T)$, is called a *model*, and we say that V is a *labeling function* or a *valuation*. Given a model $\mathfrak{M}$, we simultaneously define a set of relations on $T\times T$ and the interpretation of the language $\mathsf{PDL}_{\mathsf{tree}}(A)$ on $\mathfrak{M}$:

$$\begin{array}{rclrcl} R_{\mathtt{up}} & = & R_{\mathtt{down}}^{-1} & R_{\pi\cup\pi'} & = & R_\pi \cup R_{\pi'} \\ R_{\mathtt{left}} & = & R_{\mathtt{right}}^{-1} & R_{\pi;\pi'} & = & R_\pi \circ R_{\pi'} \\ R_{\pi^*} & = & R_\pi^* & R_{\phi?} & = & \{(t,t) \mid \mathfrak{M}, t \models \phi\} \end{array}$$

$$\begin{array}{rll} \mathfrak{M}, t \models p & \text{if, and only if,} & t \in V(p), \text{ for all } p \in \mathrm{A} \\ \mathfrak{M}, t \models \top & \text{if, and only if,} & t \in T \\ \mathfrak{M}, t \models \neg\phi & \text{if, and only if,} & \mathfrak{M}, t \not\models \phi \\ \mathfrak{M}, t \models \phi \wedge \psi & \text{if, and only if,} & \mathfrak{M}, t \models \phi \text{ and } \mathfrak{M}, t \models \psi \\ \mathfrak{M}, t \models \langle\pi\rangle\phi & \text{if, and only if,} & \exists t' \, (tR_\pi t' \text{ and } \mathfrak{M}, t' \models \phi). \end{array}$$

If $\mathfrak{M}, t \models \phi$, then we say ϕ is *satisfied* in $\mathfrak{M}$ at t. For any formula ϕ, if there is a model $\mathfrak{M}$ such that $\mathfrak{M}, \mathsf{root} \models \phi$, then we say that ϕ is *satisfiable*. For Γ a set of formulas, and ϕ a formula, we say that ϕ is a consequence of Γ (denoted by $\Gamma \models \phi$) if for every model in which Γ is satisfied at every node, ϕ is also satisfied at every node.

Note that we could have defined $\mathsf{PDL_{tree}}$ by taking `down` and `right` as the sole primitive programs and closing the set of programs under converses. As converse commutes with all program operators, these two definitions are equally expressive.

Let us consider some examples: if universally true, (1) says that every a node has a b and a c daughter, in that order, and no other daughters; and (2) says that every a node has a b first daughter followed by some number of c daughters, and no other daughters.

$$a \quad\rightarrow\quad \langle\mathtt{down}\rangle(\neg\langle\mathtt{left}\rangle\top \wedge b \wedge \langle\mathtt{right}\rangle(c \wedge \neg\langle\mathtt{right}\rangle\top)) \tag{1}$$

$$a \quad\rightarrow\quad \langle\mathtt{down}\rangle(\neg\langle\mathtt{left}\rangle\top \wedge b \wedge \langle(\mathtt{right}; c?)^*\rangle\neg\langle\mathtt{right}\rangle\top). \tag{2}$$

Now consider (3). This projects a label p down to some leaf node:

$$\langle(p?; \mathtt{down})^*\rangle(p \wedge \neg\langle\mathtt{down}\rangle\top) \tag{3}$$

That is, whenever this formula is satisfied in some model at some point t, there will be a path from t to some leaf node l such that every node on the path is marked p. We end the short examples with a list of useful abbreviations: root is short for $\neg\langle\mathtt{up}\rangle\top$, leaf is short for $\neg\langle\mathtt{down}\rangle\top$, first is short for $\neg\langle\mathtt{left}\rangle\top$, and last abbreviates $\neg\langle\mathtt{right}\rangle\top$.

3. Complexity

There are two natural problems for which we want to know the complexity. First the model checking problem: given a tree $\mathfrak{M}$, a node t, and a formula ϕ, how difficult is it to decide whether $\mathfrak{M}, t \models \phi$?

THEOREM 1 ([ALE 00]). — *$\mathfrak{M}, t \models \phi$ can be determined in time linear in the size of $\mathfrak{M}$ and of ϕ.*

See [ALE 03] for a large number of related results.

Secondly, consider the complexity of the $\mathsf{PDL_{tree}}$ consequence problem: how difficult is it to decide whether, on finite ordered trees, $\Gamma \models \chi$, for finite Γ. Decidability of this problem follows from the interpretation of $\mathsf{PDL_{tree}}$ into $L^2_{K,P}$ [ROG 98] (see the beginning of Section 5). (The decidability of the satisfiability problem for $L^2_{K,P}$ follows, in turn, via an interpretation into Rabin's $S\omega S$.) But although this reduction yields decidability, it only gives us a non-elementary decision procedure. So what is the complexity of the consequence problem?

Let us first deal with the lower bound.

THEOREM 2 ([FIS 79, SPA 93]). — *The consequence problem for the* $\mathsf{PDL_{tree}}$ *fragment with only* `down` *is* EXPTIME-*hard*.

PROOF. — This is a corollary of the analysis of the lower bound result for PDL given by [SPA 93], based on [FIS 79]. The following fragment of PDL is EXPTIME-hard: formulas of the form $\psi \wedge [a^*]\theta$ (where ψ and θ contain only the atomic program a and no embedded modalities) that are satisfiable at the root of a finite binary tree. Identifying the program a with `down`, the result follows (because $[\mathtt{down}^*]\theta \wedge \psi$ is satisfiable at the root of a finite tree if, and only if, $\theta \not\models \mathsf{root} \rightarrow \neg\psi$). ■

For full PDL this bound is optimal. There is even a stronger result: every satisfiable PDL formula ϕ can be satisfied on a model with size exponential in the length of ϕ. However with tree-based models there is no hope for such a result for it is easy to show that:

> *For every natural number* n*, there exists a satisfiable formula of size* $\mathcal{O}(n^2)$ *which can only be satisfied on at least binary branching trees of depth at least* 2^n*.*

A formula containing most of the requirements to force such a deep branch is given in Proposition 6.51 of [BLA 01]. To this formula we only have to add the conjunct $[\mathtt{down}^*](\langle\mathtt{down}\rangle p \wedge \langle\mathtt{down}\rangle\neg p)$ for some new variable p to enforce binary branching. Note that the size of such a model is double exponential in the size of the formula. This means that a decision algorithm which tries to construct a tree model must use at least exponential space, as it will need to keep a whole branch in memory.

So we have to think more carefully about the upper bound. One way to proceed is to take a clue from the completeness proof for a related language in [BLA 94]. Instead of constructing a model it is possible to design an algorithm which searches for a "good" set of labellings of the nodes of a model. Label sets consist of subformulas of the formula ϕ whose satisfiability is to be decided. From a good set of labels we can construct a labeled tree model which satisfies ϕ. The number of labels is bound by an exponential in the number of subformulas of ϕ, and the search for a good set of labels among the possible ones can be implemented in time polynomial in the number of possible labels using the technique of elimination of Hintikka sets developed in [PRA 79]. A direct proof using this technique was given in [BLA 03] for the language $\mathcal{L}_{cp}$ (see Section 4). Unfortunately, the technique cannot be straightforwardly applied to $\mathsf{PDL_{tree}}$. Here we show how an old result of Vardi and Wolper [VAR 86] on deterministic PDL with converse yields the desired upper bound.

THEOREM 3. — *The* $\mathsf{PDL_{tree}}$ *consequence problem is in* EXPTIME.

PROOF. — First note that $\gamma_1, \ldots, \gamma_n \models \chi$ if and only if $\models \mathsf{root} \rightarrow ([\mathtt{down}^*](\gamma_1 \wedge \ldots \wedge \gamma_n) \rightarrow \chi)$. Thus we need only decide satisfiability of $\mathsf{PDL_{tree}}$ formulas at the root of finite trees.

Consider the language $\mathcal{L}_2$, the modal language with the two programs $\{\downarrow_1, \downarrow_2\}$ and their inverses $\{\uparrow_1, \uparrow_2\}$. $\mathcal{L}_2$ is interpreted on finite at most *binary-branching* trees, with $\downarrow_1$ and $\downarrow_2$ interpreted by the first and second daughter relation, respectively. We will effectively reduce $\mathsf{PDL_{tree}}$ satisfiability to $\mathcal{L}_2$ satisfiability. $\mathcal{L}_2$ is a fragment of deterministic PDL with converse. [VAR 86] shows that the satisfiability problem for this latter language is decidable in EXPTIME over the class of all models. This is done by constructing for each formula ϕ a tree automaton A_ϕ which accepts exactly all tree models in which ϕ is satisfied. Thus deciding satisfiability of ϕ reduces to checking emptiness of A_ϕ. The last check can be done in time polynomial in the size of A_ϕ. As the size of A_ϕ is exponential in the length of ϕ, this yields the exponential time decision procedure.

But we want satisfiability on *finite* trees. This is easy to cope with in an automata-theoretic framework: construct an automaton $A_{\mathit{fin\_tree}}$, which accepts only finite binary trees, and check emptiness of $A_\phi \cap A_{\mathit{fin\_tree}}$. The size of $A_{\mathit{fin\_tree}}$ does not depend on ϕ, so this problem is still in EXPTIME.

The reduction from $\mathsf{PDL_{tree}}$ to $\mathcal{L}_2$ formulas is very simple: replace the $\mathsf{PDL_{tree}}$ programs $\mathtt{down}, \mathtt{up}, \mathtt{right}, \mathtt{left}$ by the $\mathcal{L}_2$ programs

$$\downarrow_1;\downarrow_2^*,\quad \uparrow_2^*;\uparrow_1,\quad \downarrow_2,\quad \uparrow_2,$$

respectively. It is straightforward to prove that this reduction preserves satisfiability, following the reduction from $S\omega S$ to $S2S$ as explained in [WEY 02]: a $\mathsf{PDL_{tree}}$ model $(T, R_{\mathtt{right}}, R_{\mathtt{down}}, V)$ is turned into an $\mathcal{L}_2$ model (T, R_1, R_2, V) by defining

$$R_1 = \{(x, y) \mid xR_{\mathtt{down}}y \text{ and } y \text{ is the first daughter of } x\}$$

and $R_2 = R_{\mathtt{right}}$. Turn an $\mathcal{L}_2$ model (T, R_1, R_2, V) into a $\mathsf{PDL_{tree}}$ model $(T, R_{\mathtt{right}}, R_{\mathtt{down}}, V)$ by defining $R_{\mathtt{right}} = R_2$ and $R_{\mathtt{down}} = R_1 \circ R_2^*$. ■

4. Expressivity 1: first-order logic

Let $\mathcal{L}_{FO}$ denote the first-order language over the signature with binary predicates $\{R_{\mathtt{down}^+}, R_{\mathtt{right}^+}\}$ and countably many unary predicates. $\mathcal{L}_{FO}$ is interpreted on ordered trees in the obvious way: $R_{\mathtt{down}^+}$ is interpreted by the transitive closure of the daughter-of relation, and $R_{\mathtt{right}^+}$ is interpreted by the transitive closure of the right-sister relation. Note that the language is first order, even though we interpret the two primitive relations as second order relations over a more primitive relations. This is not strange, but just another perspective: we take descendant as primitive instead of the immediate daughter relation. Of course the latter is first order definable from the descendant relation.

Two other modal languages proposed in the model-theoretic syntax literature can be considered as first-order fragments of $\mathsf{PDL_{tree}}$. That is, they can be considered as versions of $\mathsf{PDL_{tree}}$ with a more limited repertoire of programs. As first-order logic is a natural point of reference for the expressivity of languages it is useful to consider first-order fragments of $\mathsf{PDL_{tree}}$. We consider two, one predating and one postdating the introduction of $\mathsf{PDL_{tree}}$.

The language proposed by Blackburn, Meyer-Viol and de Rijke [BLA 96], here called $\mathcal{L}_{Core}$, contains only the core machinery for describing trees: the four basic programs plus their *transitive* closures, denoted by a superscript $(\cdot)^+$. This language is precisely as expressive[1] as (i.e., can define the same sets of nodes as) the fragment of $\mathsf{PDL_{tree}}$ generated by the following programs:

$$\pi ::= \texttt{left} \mid \texttt{right} \mid \texttt{up} \mid \texttt{down} \mid \pi^*,$$

or equivalently by

$$\begin{aligned}\pi \quad ::= \quad & \texttt{left} \mid \texttt{right} \mid \texttt{up} \mid \texttt{down} \mid \pi;\pi \mid \pi \cup \pi \mid \phi? \mid \\ & a^*, \text{ for } a \text{ one of the four atomic programs.}\end{aligned}$$

The language proposed by Palm [PAL 99], here called $\mathcal{L}_{cp}$ (with cp abbreviating *conditional path*), lies between $\mathcal{L}_{Core}$ and $\mathsf{PDL_{tree}}$ with respect to expressive power. It can be thought of as the fragment of $\mathsf{PDL_{tree}}$ generated by the following programs:

$$\pi ::= \texttt{left} \mid \texttt{right} \mid \texttt{up} \mid \texttt{down} \mid \pi;\phi? \mid \pi^*,$$

or equivalently by

$$\begin{aligned}\pi \quad ::= \quad & \texttt{left} \mid \texttt{right} \mid \texttt{up} \mid \texttt{down} \mid \pi;\pi \mid \pi \cup \pi \mid \phi? \mid \\ & (a;\phi?)^*, \text{ for } a \text{ one of the four atomic programs.}\end{aligned}$$

Note that while the two definitions for $\mathcal{L}_{cp}$ give rise to equally expressive languages, not every program of the second language is equivalent to a program of the first language. For example, the programs $(a;\phi?)^+$ and $(\phi?;a)^*$ can be expressed only in the second language. In this paper we will consider $\mathcal{L}_{cp}$ to be the fragment of $\mathsf{PDL_{tree}}$ generated by the programs given in the second definition.

Both languages are easily seen to be fragments of $\mathcal{L}_{FO}$, the first order language for sibling ordered trees. In fact we know exactly which fragments. A *tree property* is a class of pairs (T, N) consisting of a tree T and a subset N of its domain. A tree property P is *definable* in a language L if there is a formula $\phi \in L$ such that (T, N) is in P if, and only if, the denotation of ϕ in T equals N. For instance, the property of having at least two children is definable by the formula $\langle\texttt{down}\rangle\langle\texttt{right}\rangle\top$.

THEOREM 4 ([PAL 97, MAR 04A]). — *The following are equivalent on ordered trees. For P a tree property:*

– *P is definable by an $\mathcal{L}_{cp}$ formula;*

– *P is definable by an $\mathcal{L}_{FO}$ formula in one free variable.*

1. At this point we are only interested in the expressiveness of the modal language, not of the set of programs. So we measure the expressive power of a $\mathsf{PDL_{tree}}$ fragment in terms of which sets of nodes can be defined in it.

THEOREM 5 ([MAR 04B]). — *The following are equivalent on ordered trees. For P a tree property:*

– *P is definable by an $\mathcal{L}_{Core}$ formula;*

– *P is definable by an $\mathcal{L}_{FO}$ formula in one free variable which*

1) contains at most two (free and bound) variables (possibly reused), and

2) which may use additional atomic relations corresponding to the daughter-of and right-sister relation.

The first theorem can be seen as a generalization of Kamp's Theorem [KAM 68] to ordered trees. The theorem was announced in [PAL 97], but the proof is hard to follow. [MAR 04a] contains a proof based on Gabbay's notion of separation [GAB 84]. The second theorem is also a generalization of a result for temporal logic on linear structures, this time due to Etessami, Vardi and Wilke [ETE 97].

We end this section by giving some insight into the expressive power of $\mathcal{L}_{cp}$. First note that the temporal until(ϕ, ψ) modality can be expressed, in all four directions. For the downward direction, until(ϕ, ψ) is expressed as $\langle(\psi?;\mathtt{down})^*\rangle\phi$. Indeed, this formula is true at a node n if, and only if, ϕ is true at n or there exists a descendant n' of n at which ϕ is true and at all nodes, starting with n and descending to n' exclusive, ψ is true.

So far, we have only considered expressivity with respect to sets of nodes. We will now consider expressivity with respect to binary relations on the set of nodes: which binary relations can be defined by means of a program of the language?

THEOREM 6 ([MAR 05]). — *The following are equivalent on ordered trees. For P a binary relation:*

– *P is definable by an $\mathcal{L}_{cp}$ program;*

– *P is definable by an $\mathcal{L}_{FO}$ formula in two free variables.*

We now give a representative example of a first-order formula and its equivalent $\mathcal{L}_{cp}$ program. In the context of XML documents, the order in which the nodes are written is an important relation, called *document order*. Figure 1 contains an example of an XML file, its corresponding tree model and the numbers of the nodes correspond to their document ordering. The document order relation $\ll$ is defined as

$$\ll \;\equiv\; \mathtt{down}^+ \,\cup\, \mathtt{up}^*;\mathtt{right}^+;\mathtt{down}^*.$$

On finite trees it makes sense to speak about the successor relation of the document order. The simple definition is $\ll \cap \overline{\ll \circ \ll}$. It can be defined also with the $\mathcal{L}_{cp}$ programs as

$$\mathtt{down};\mathsf{first}? \,\cup\, \mathsf{leaf}?;\mathtt{right} \,\cup\, (\mathsf{last}?;\mathtt{up})^+;\mathtt{right}. \qquad (4)$$

Next we show how to define the relation

$$x \ll y \wedge \phi(x) \wedge \psi(y) \wedge \forall z(x \ll z \ll y \rightarrow \phi(z)), \qquad (5)$$

```
<AA1>
  <BB2>
     <CC3/>
     <CC4/>
  </BB2>
  <DD5>
     <EE6/>
     <FF7/>
  </DD5>
  <GG8>
     <HH9/>
     <JJ10/>
  </GG8>
</AA1>
```

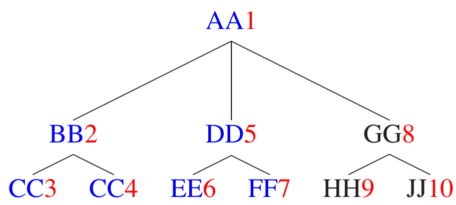

Figure 1. *An XML document and its corresponding tree model*

from the $\mathcal{L}_{cp}$ programs, and once we have this, the "until in document-order" modality: $\text{until}_{\ll}(\psi, \phi)$ holds at x if, and only if, $\psi(x) \vee \exists y(5)$. Note that Theorem 4 ensures that the set $\psi(x) \vee \exists y$ (5) is $\mathcal{L}_{cp}$ definable, but not that the *relation* (5) is definable from the $\mathcal{L}_{cp}$ programs.

We must use the definition of $\mathcal{L}_{cp}$ programs containing union and composition. The definition is a case distinction based on the definition of $x \ll y$:

1) $x\,\texttt{down}^+\ y$
2) $x\,\texttt{up}^+;\texttt{right}^+;\texttt{down}^+\ y$
3) $x\,\texttt{up}^+;\texttt{right}^+\ y$
4) $x\,\texttt{right}^+;\texttt{down}^+\ y$
5) $x\,\texttt{right}^+\ y$.

We only show the easiest (first) and the hardest (second) case. The others are variations of these. For the first case we want to express that

$$x\ \texttt{down}^+ y \wedge \phi(x) \wedge \psi(y) \wedge \forall z(x \ll z \ll y \rightarrow \phi(z)).$$

We explain our formulas by examples. Suppose x is node 1 and y is node 7 in Figure 1. Then ϕ must hold at nodes 1–6 and ψ must hold at node 7. This holds just in case x and y are related by

$$\begin{aligned} \phi? \quad &; \\ (\texttt{down}; (\phi \wedge [\texttt{left}^+][\texttt{down}^*]\phi)?)^* \quad &; \\ \texttt{down}; (\psi \wedge [\texttt{left}^+][\texttt{down}^*]\phi)?. & \end{aligned} \tag{6}$$

The first line of (6) ensures that ϕ holds at the node 1. The second line is evaluated at the node 5 and ensures that the nodes 2–5 make ϕ true. The third line says that ψ holds at the node 7 and ϕ holds at the node 6.

For the second case we want to express that

$$x \ \mathtt{up}^+ ; \mathtt{right}^+ ; \mathtt{down}^+ y \wedge \phi(x) \wedge \psi(y) \wedge \forall z(x \ll z \ll y \rightarrow \phi(z)). \qquad (7)$$

This holds exactly when x and y are related by

$$[\mathtt{down}^*]\phi? \quad ; \qquad (8)$$

$$([\mathtt{right}^+][\mathtt{down}^*]\phi?; \mathtt{up})^+ \quad ; \qquad (9)$$

$$(\mathtt{right}; [\mathtt{down}^*]\phi?)^* \quad ; \qquad (10)$$

$$\mathtt{right}; \phi? \quad ; \qquad (11)$$

$$(\mathtt{down}; (\phi \wedge [\mathtt{left}^+][\mathtt{down}^*]\phi)?)^* \quad ; \qquad (12)$$

$$\mathtt{down}; (\psi \wedge [\mathtt{left}^+][\mathtt{down}^*]\phi)?. \qquad (13)$$

This formula is best explained using a more elaborate tree, as in Figure 2. Suppose nodes C and R stand in the relation (7). Then (8) ensures that $\{A, B, C\}$ makes ϕ true; the test $[\mathtt{right}^+][\mathtt{down}^*]\phi$ in (9) will be evaluated at nodes C and G, thereby ensuring that ϕ holds in $\{F, D, E\}$ and $\{J, H, I\}$, respectively. The test $[\mathtt{down}^*]\phi$ in (10) will be evaluated at all nodes strictly in between K and U, so here taking care that ϕ holds at $\{N, L, M\}$. (11) ensures that ϕ is true at U. Now (12) and (13) are just the subprograms of (6) from the first case, ensuring that ϕ holds at $\{Q, O, P, T\}$ and ψ holds at $\{R\}$.

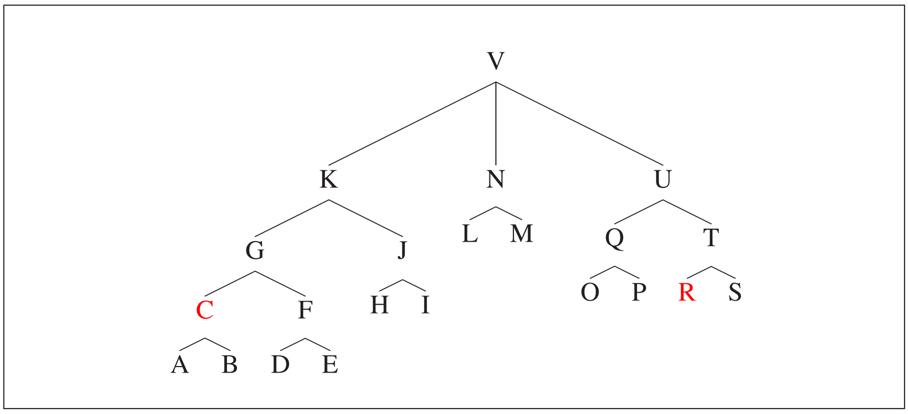

Figure 2. *Example tree for the second case*

5. Expressivity 2: second-order properties

In this section we look at three concrete examples of non-trivial second-order properties of trees that are expressible in $\mathsf{PDL}_{\mathsf{tree}}$; first though, some background. The language $\mathsf{PDL}_{\mathsf{tree}}$ can express properties beyond the reach of $\mathcal{L}_{FO}$. For example, $\mathsf{PDL}_{\mathsf{tree}}$ can express the property of having an odd number of daughters:

$$\langle \mathtt{down} \rangle (\mathsf{first} \wedge \langle (\mathtt{right}; \mathtt{right})^* \rangle \mathsf{last}). \tag{14}$$

Note that the second conjunct $\langle (\mathtt{right}; \mathtt{right})^* \rangle \mathsf{last}$ says that by chaining together a succession of double `right` steps we can reach the rightmost daughter node — which means that there must be an odd number of daughter nodes. This is a property that no $\mathcal{L}_{FO}$ formula can express.

On the other hand, $\mathsf{PDL}_{\mathsf{tree}}$ is contained in $L^2_{K,P}$, Rogers monadic second-order logic of variably branching trees [ROG 98]. $L^2_{K,P}$ just extends $\mathcal{L}_{FO}$ by quantification over unary predicates. The translation of $\mathsf{PDL}_{\mathsf{tree}}$ formulas into $L^2_{K,P}$ is straightforward. Note, in particular, that we can use second-order quantification to define the transitive closure of a relation: for R any binary relation, xR^*y holds iff

$$x = y \vee \forall X (X(x) \wedge \forall z, z' (X(z) \wedge zRz' \to X(z')) \to X(y)).$$

Thus $\mathsf{PDL}_{\mathsf{tree}}$ can be seen as a fragment of unary $L^2_{K,P}$. Kracht showed that the inclusion is strict:

THEOREM 7 ([KRA 99]). — *Unary $L^2_{K,P}$ is strictly more expressive than* $\mathsf{PDL}_{\mathsf{tree}}$.

This brings us to the most important open problem concerning $\mathsf{PDL}_{\mathsf{tree}}$:

OPEN PROBLEM. — Characterize the expressive power of $\mathsf{PDL}_{\mathsf{tree}}$ interpreted on finite ordered trees in terms of a suitable fragment of monadic second-order logic. □

Within the context of query languages for XML documents a number of proposals for second-order languages have been made. The goal, then, is to express unary MSO, MSO formulas denoting a set of nodes. We mention monadic datalog of [GOT 02] and the efficient tree logic of [NEV 00a], which are both as expressive as unary MSO.

Neven and Schwentick [NEV 00a] argue that unary MSO rather than $\mathcal{L}_{FO}$ is the gold standard for a language designed for specifying nodes in finite ordered trees. Their most convincing example is a variant of the boolean circuit problem. In order to obtain a better understanding of the second order expressivity of $\mathsf{PDL}_{\mathsf{tree}}$, we encode a number of second-order properties in $\mathsf{PDL}_{\mathsf{tree}}$. In addition to the boolean circuit problem just mentioned, we encode the frontier problem and we show that finiteness of ordered trees can be expressed in $\mathsf{PDL}_{\mathsf{tree}}$. The frontier problem is a typical linguistic problem. Expressing finiteness within a large class of tree-like structures shows the robustness of the language. We look at the upshot of these examples at the end of this section. We start with the frontier problem.

5.1. *The frontier problem*

The *frontier* of a tree is the set of leaves ordered from left to right. In a parse tree of a natural language sentence, the frontier is exactly that sentence. Usually the frontier is where the actual data contained in a tree is located.

Given a condition ϕ on the frontier, we want to write an $\mathsf{PDL_{tree}}$ expression which is true at the root of a tree if, and only if, the frontier of the tree satisfies ϕ. For instance, ϕ could be a regular expression over atomic symbols, like $(p; q)^*$. The most natural application is when we know that each leaf node makes exactly one atomic symbol true. Then a tree satisfies ϕ if and only if the frontier is a word in $(p; q)^*$. But nothing forbids us to use arbitrary complex $\mathsf{PDL_{tree}}$ formulas in place of p and q. E.g., $\langle \mathtt{up}^* \rangle \mathtt{np}$ states that the current word of the parsed sentence is part of a noun-phrase ("an np"). Thus we do not view the frontier as a unique string, but as an infinite collection of strings, made up from formulas which are true at the respective nodes. Now let r be a regular expression in which arbitrary $\mathsf{PDL_{tree}}$ formulas are the letters. We say that a tree's frontier $l_1 \ldots, l_n$ satisfies r if, and only if, there are $\mathsf{PDL_{tree}}$ formulas ϕ_i such that for all $i, l_i \models \phi_i$ and the string $\phi_1, \ldots, \phi_n$ is a word in r.

What we need for expressing frontier conditions is the successor relation between frontier nodes. This is naturally defined using the document order relation from the previous section. A frontier node y is the successor of a frontier node x if and only if $x \ll y$ and there is no leaf node in between x and y in the document order. An intuitive definition of the next_frontier_node relation between leaves can now be given as:

$$\mathsf{leaf}?; [(\neg\mathsf{last})? \cup (\mathsf{last}?; \mathtt{up})^+]; \mathtt{right}; (\mathtt{down}; \mathsf{first}?)^*; \mathsf{leaf}?. \tag{15}$$

Because we evaluate the $\mathsf{PDL_{tree}}$ formula at the root, we should add to (15) the step from the root to the first leaf. So define the `next_frontier_node` relation as

$$\mathsf{root}?; (\mathtt{down}; \mathsf{first}?)^*; \mathsf{leaf}? \quad \cup \quad (15).$$

Let $\mathsf{last\_frontier\_node}$ be an abbreviation of $\mathsf{leaf} \wedge \langle(\mathsf{last}?; \mathtt{up})^*\rangle \mathsf{root}$, which indeed is true exactly at the last frontier node (or simply at the root, if the root is the only node in the model).

Now let r be a regular expression over a set of $\mathsf{PDL_{tree}}$ formulas. Then for any tree T, T's frontier satisfies r if and only if the root of T satisfies $\langle r^\circ \rangle \mathsf{last\_frontier\_node}$, where r° is r with ; placed between all $\mathsf{PDL_{tree}}$ formulas which act as letters in r and any such formula ϕ is replaced by $\mathtt{next\_frontier\_node}; \phi?$. For instance, the frontier is in $(ab)^*$, where a and b are atomic symbols if, and only if, the root satisfies

$$\langle (\mathtt{next\_frontier\_node}; a?; \mathtt{next\_frontier\_node}; b?)^* \rangle \mathsf{last\_frontier\_node}.$$

Note that the formula is true on a tree containing only the root; thus it correctly recognizes the empty string.

5.2. *The boolean circuit problem*

We show how the boolean circuit problem can be expressed in $\mathsf{PDL}_{\mathsf{tree}}$. Our $\mathsf{PDL}_{\mathsf{tree}}$ formula is based on the same idea as in [NEV 00b]: use a depth first traversal of the tree. We start with defining the boolean circuit problem.

DEFINITION 8 (BOOLEAN CIRCUITS). — Boolean circuits *are finite* $\{1, 0, \mathrm{C}, \mathrm{D}\}$-*labeled ordered binary trees such that*

1) each leaf is labeled with exactly one of $\{1, 0\}$*, and*

2) each non-leaf is labeled with exactly one of $\{\mathrm{C}, \mathrm{D}\}$.

If $\mathcal{B}$ is a boolean circuit and $b \in \mathcal{B}$ then with $\mathcal{B}_b$ we denote the subtree of $\mathcal{B}$ rooted at b. With $\mathcal{B}_{\overline{b}}$ we denote the tree which we obtain by removing everything below b. So in particular we have that b is a leaf of $\mathcal{B}_{\overline{b}}$.

The intended meaning of the labels is as one might expect: 1 means 'true', 0 means 'false', C means conjunction and D means disjunction. For any boolean circuit $\mathcal{B}$, define the boolean function eval from the domain of $\mathcal{B}$ to $\{$'true', 'false'$\}$ in the expected way. For instance, as the Datalog program:

```
eval(x)   :-   1(x).
eval(x)   :-   D(x), R_down(x,y), eval(y).
eval(x)   :-   C(x), R_down(x,y), R_right(y,z), eval(y), eval(z).
```

Also for any $b \in \mathcal{B}$ let height(b) denote the length of the longest path starting at, but not including, b to a leaf. So if b is a leaf, then height(b) $= 0$.

GENERAL IDEA. — To check if a boolean circuit evaluates to true we look for substructures that can be constructed as follows. We start at the root and move down. At disjunctive nodes we select one child. At conjunctive nodes we take both children. When we reach a leaf, it should be labeled with 1. We check if such a substructure exists in a depth first fashion. So, we walk down the tree, where at conjunctive nodes we always take the left route and make sure (by selecting the correct child at disjunctive nodes) we end up in a leaf labeled 1. We let the relation R_0 denote such a path. That is, for all x and y we have xR_0y if, and only if, the following three cases apply.

1) $\exists\, k \geq 1\; t_1, \ldots, t_k$ s.t. $x = t_1 \mathtt{down} t_2 \mathtt{down} \cdots \mathtt{down} t_k = y$

2) For all $1 \leq i < k$ if $t_i \Vdash \mathrm{C}$, then $t_{i+1} \Vdash \mathsf{first}$

3) $t_k \Vdash 1$

Next we walk up again until we are at a left child of a conjunctive node. We move right, to node b_r say, and repeat the procedure. When we return at node b_r we realize that we are about to enter a conjunctive node from the right and move further up until the next conjunctive node. With R_1 we denote this relation. So for all x and y we have xR_1y if, and only if, the following two cases apply.

1) $\exists k \geq 1\; t_1, \ldots, t_k$ s.t. $x = t_1 \mathtt{up} t_2 \mathtt{up} \cdots \mathtt{up} t_k = y$

2) For all $1 \leq i < k, t_i \Vdash \langle \mathtt{up} \rangle \mathrm{C} \rightarrow \mathsf{last}$

When we reach the root of the boolean circuit the procedure stops. We can express both relations R_0 and R_1 as $\mathsf{PDL_{tree}}$ programs π_0 and π_1 as follows. Let π_0 be the program, which corresponds to R_0. That is

$$\pi_0 = ((\mathrm{D}?; \mathtt{down}) \cup (\mathrm{C}?; \mathtt{down}; \mathsf{first}?))^*; 1?.$$

Let π_1 be the program corresponding to R_1. That is

$$\pi_1 = ((\langle \mathtt{up} \rangle \mathrm{C} \rightarrow \mathsf{last})?; \mathtt{up})^*.$$

Finally define

$$\beta = \langle \pi_0; \pi_1; (\mathtt{right}; \pi_0; \pi_1)^* \rangle \mathsf{root}.$$

Before we move on let us make a remark. On first sight one might think that we need in the definition of R_1 a third clause. Namely

3) $t_k \Vdash \langle \mathtt{up} \rangle \mathrm{C} \wedge \neg \mathsf{last}$ or $t_k \Vdash \mathsf{root}$.

And, consequently, instead of π_1 we should have

$$\pi_1; (\langle \mathtt{up} \rangle \mathrm{C} \wedge \neg \mathsf{last} \vee \mathsf{root})?.$$

This is not necessary. With the current definition of R_1 we allow for a check (but do not consider it necessary) that the second child of a disjunctive node is true when we already know that the first child is. This is just as harmless as it is useless. Nevertheless, the proof below (in particular Lemma 13) does not work without this omission. □

THEOREM 9. — *β is forced at the root r of a boolean circuit iff eval(r) is true.*

The proof follows below, but first several lemmas.

LEMMA 10. — *Let $\mathcal{B}$ be a boolean circuit. For all nodes $b \in \mathcal{B}$ we have the following.*

1) $b \Vdash \beta \wedge \mathrm{C} \rightarrow [\mathtt{down}]\beta$

2) $b \Vdash \beta \wedge \mathrm{D} \rightarrow \langle \mathtt{down} \rangle \beta$

PROOF. — First we show 1. Suppose $b \Vdash \beta \wedge \mathrm{C}$. Let b_l be the left child of b and b_r be the right child of b. It is easy to see that $b_l \Vdash \beta$. To show that $b_r \Vdash \beta$ we need a lemma.

LEMMA 11. — *For any x for which not $x(\mathtt{up})^* b_l$ (e.g. $x \notin \mathcal{B}_{b_l}$) we have that if $b_l(\pi_0; \pi_1; (\mathtt{right}; \pi_0; \pi_1)^*)x$ then $b_l(\mathtt{right}; \pi_0; \pi_1)^* x$.*

PROOF. — Choose x as stated. We show with induction on n that

$$\text{if } b_l(\pi_0; \pi_1; (\mathtt{right}; \pi_0; \pi_1)^n)x \text{ then } b_l(\mathtt{right}; \pi_0; \pi_1)^* x.$$

If $n = 0$ then for some t, $b_l \pi_0 t \pi_1 x$. Clearly $t(\mathtt{up})^* b_l$ and $t(\mathtt{up})^* x$. So, by choice of x, $b_l(\mathtt{up})^+ x$. But this is clearly in contradiction with the definition of π_1.

Now suppose $b_l(\pi_0; \pi_1; (\mathtt{right}; \pi_0; \pi_1)^{n+1})x$. Choose t such that

$$b_l(\pi_0; \pi_1; (\mathtt{right}; \pi_0; \pi_1)^n)t \text{ and } t(\mathtt{right}; \pi_0; \pi_1)x.$$

We can assume that $t(\mathtt{up})^* b_l$ (otherwise we are done by (IH)). We also can assume that $t \neq b_l$ and thus $t(\mathtt{up})^+ b_l$. Fix some t' for which $t(\mathtt{right}; \pi_0)t' \pi_1 x$. By the above we obtain $t'(\mathtt{up})^+ b_l$. Similar as in the case $n = 0$ this leads us to a contradiction. ■

Now we continue with showing that $b_r \Vdash \beta$. Since $b_l \Vdash \beta$ we can find some $x_1, x_2, \ldots$ such that

$$b_l = x_1(\pi_0; \pi_1)x_2(\mathtt{right}; \pi_0; \pi_1)x_3 \cdots (\mathtt{right}; \pi_0; \pi_1)r,$$

where r is the root of $\mathcal{B}$. Let i be that smallest number such that not $x_i(\mathtt{up})^* b_l$. Note that $i > 2$. So, by the above lemma and by choice of i, we have $b_l(\mathtt{right}; \pi_0; \pi_1)x_i$. So, $b_r(\pi_0; \pi_1)x_i$ and thus $b_r \Vdash \beta$. We have shown 1.

Item 2 is rather trivial. For if we suppose that $b \Vdash \beta \wedge \mathrm{D}$ then it is easy to verify, using the definition of π_0, that $b \Vdash \langle \mathtt{down} \rangle \beta$. ■

COROLLARY 12. — *Let $\mathcal{B}$ be a boolean circuit. For all nodes $b \in \mathcal{B}$ we have that if $b \Vdash \beta$ then eval(b) is true.*

PROOF. — Induction on height(b). If height(b) $= 0$ then the claim is clear by the definition of π_0. So suppose height(b) > 0. There are two cases to consider.

Case: $b \Vdash$ C. By Lemma 10 we have $b \Vdash [\mathtt{down}]\beta$. So, by (IH), we have that for all children b' of b that eval(b') is true and thus eval(b) is true.

Case: $b \Vdash$ D. By Lemma 10 we have $b \Vdash \langle \mathtt{down} \rangle \beta$. So, by (IH), for some child b' of b we have eval(b') is true and thus eval(b) is true. ■

LEMMA 13. — *Let $\mathcal{B}$ be a boolean circuit. For all $b \in \mathcal{B}$ for which eval(b) is true we have that $b(\pi_0; \pi_1; (\mathtt{right}; \pi_0; \pi_1)^*)b$.*

PROOF. — Induction on height(b). If height(b) $= 0$ then this is clear. So suppose height(b) > 0. There are two cases to consider.

Case: $b \Vdash$ C. Then for b's children b_l and b_r we have that eval(b_l) and eval(b_r) are true. By (IH), $b_l(\pi_0; \pi_1; (\mathtt{right}; \pi_0; \pi_1)^*)b_l$ and $b_r(\pi_0; \pi_1; (\mathtt{right}; \pi_0; \pi_1)^*)b_r$. So, the pair (b, b) is contained in the following relation:

$$\mathrm{C}?; \mathtt{down}; \mathsf{first}?; \pi_0; \pi_1; (\mathtt{right}; \pi_0; \pi_1)^*; \mathtt{right}; \pi_0; \pi_1; (\mathtt{right}; \pi_0; \pi_1)^*; \mathsf{last}?; \mathtt{up}.$$

Thus, as one can easily verify, we have

$$b(\pi_0; \pi_1; (\mathtt{right}; \pi_0; \pi_1)^*)b.$$

Case: $b \Vdash$ D. Then for at least one child b_i of b we have $\mathrm{eval}(b_i)$ is true. So by (IH) we obtain $b_i(\pi_0; \pi_1; (\mathtt{right}; \pi_0; \pi_1)^*)b_i$. Thus

$$b(\mathrm{D}?; \mathtt{down}; (\pi_0; \pi_1; (\mathtt{right}; \pi_0; \pi_1)^*); (\neg\langle\mathtt{up}\rangle\mathrm{C})?; \mathtt{up})b.$$

Which implies $b(\pi_0; \pi_1; (\mathtt{right}; \pi_0; \pi_1)^*)b$. ■

Now we are ready to prove Theorem 9.

PROOF (OF THEOREM 9). — ($\Rightarrow$) Immediate from Corollary 12. ($\Leftarrow$) Suppose $\mathrm{eval}(r)$ is true. By Lemma 13 we have $r(\pi_0; \pi_1; (\mathtt{right}; \pi_0; \pi_1)^*)r$. So in particular $r \Vdash \langle\pi_0; \pi_1; (\mathtt{right}; \pi_0; \pi_1)^*\rangle\mathsf{root}$, i.e., $r \Vdash \beta$. ■

5.3. *Expressing finiteness*

We let go of the restriction to finite trees. Normally one would define arbitrary trees as partially ordered sets $(W, <)$ with a unique root and such that for each $w \in W$ the set $\{v \mid v < w\}$ is well-ordered By $<$. The height of a node w is then defined as the order-type of $\{v \mid v < w\}$ and we say that a tree is of height ω when the height of each node is finite. We can do a little bit better. Below we define first-order definable structures such that the part that $\mathsf{PDL}_{\mathsf{tree}}$ can see is a tree of height ω.

First, for a binary relation R we say that y is a *direct successor* of x when xRy and for no z we have $xRzRy$. We define *direct predecessor* in a similar way. We say that R is *discrete* when for any xRy such that y is not a direct successor of x, there exists some direct successor z of x with $xRzRy$. Notice that discrete relations are always irreflexive. We say that a structure $\langle T, R_{\mathtt{down}^+}, R_{\mathtt{right}^+}\rangle$ (note that, in this context, $R_{\mathtt{down}^+}$ and $R_{\mathtt{right}^+}$ are primitive relation symbols themselves) is *tree-like* when

1) $R_{\mathtt{down}^+}$ is a discrete and partial order on T with a unique root,

2) each $t \in T$ has at most one direct $R_{\mathtt{down}^+}$-predecessor,

3) $R_{\mathtt{right}^+}$ is discrete and linearly orders the direct successors of any $t \in T$, in particular if $xR_{\mathtt{right}^+}y$ then x and y have the same direct $R_{\mathtt{down}^+}$-predecessor.

Clearly, this class of structures is first-order definable within the class of Kripke frames with two accessibility relations and any tree is a tree-like structure. We define the relations $R_{\mathtt{down}}$, $R_{\mathtt{right}}$ and all the other relations R_π that may occur within $\mathsf{PDL}_{\mathsf{tree}}$-modalities as above in Section 2. But note that although we do have $R^+_{\mathtt{down}} \subseteq R_{\mathtt{down}^+}$, $R^+_{\mathtt{right}} \subseteq R_{\mathtt{right}^+}$, in general these inclusions will be proper. If $\mathcal{T} = \langle T, R_{\mathtt{down}^+}, R_{\mathtt{right}^+}\rangle$ is a tree-like structure with root r, then we write $\mathcal{T}_r$ for the structure $\langle r\rangle_{\langle T, R_{\mathtt{down}}, R_{\mathtt{right}}\rangle}$, the substructure of $\langle T, R^+_{\mathtt{down}}, R^+_{\mathtt{right}}\rangle$ generated by r using the defined relations $R_{\mathtt{down}}$ and $R_{\mathtt{right}}$, in the usual modal logic sense. Of course for any $\mathsf{PDL}_{\mathsf{tree}}$ formula ϕ we have that $\mathcal{T}, r \Vdash \phi$ iff $\mathcal{T}_r, r \Vdash \phi$. So without danger of confusion we can write $r \Vdash \phi$.

As a corollary to the proof of the definability of boolean circuits we will show that $\mathsf{PDL}_{\mathsf{tree}}$ can define finiteness of tree-like structures.

THEOREM 14. — *There exists a* $\mathsf{PDL}_{\mathsf{tree}}$*-formula* Fin *such that for any tree-like structure* $\mathcal{T}$ *with root* r *we have* $\mathcal{T}, r \Vdash \mathrm{Fin}$ *if, and only if,* $\mathcal{T}$ *is finite.*

PROOF. — Let δ and γ be as defined in (16) and (17) below and let Fin be $\delta \wedge \gamma$. The proof proceeds in stages. In Lemma 15 we show that it is sufficient to show that $\mathcal{T}_r, r \Vdash \mathrm{Fin}$ if, and only if, $\mathcal{T}_r$ is finite. This latter is shown in Lemmas 16, 17 and 18. ∎

LEMMA 15. — *For any tree-like structure* $\mathcal{T}$ *with root* r*,* $\mathcal{T}_r$ *is an ordered*[2] *tree of height* ω*, and* $\mathcal{T}_r$ *is finite if, and only if,* $\mathcal{T}$ *is finite.*

PROOF. — The first assertion is a direct consequence of the definition of tree-like structures. The second assertion follows from the fact that if x is a leaf in $\mathcal{T}_r$ then by discreteness there does not exist any $R_{\mathtt{down}^+}$ descendant of x in $\mathcal{T}$. ∎

As a first approximation for finiteness put

$$\delta = [\mathtt{down}^*](\langle \mathtt{left}^* \rangle \mathsf{first} \wedge \langle \mathtt{right}^* \rangle \mathsf{last}). \tag{16}$$

LEMMA 16. — *For any tree-like structure* $\mathcal{T}$ *with root* r *we have that* $\mathcal{T}_r, r \Vdash \delta$ *if, and only if,* $\mathcal{T}_r$ *is finitely branching.*

PROOF. — The left to right direction holds since if $t \in \mathcal{T}_r$ has infinitely many children then by discreteness we can find an infinite, to the left or to the right, $R_{\mathtt{right}}$-chain. The converse is obvious. ∎

So in order to define the class of finite tree-like structures it is enough to define the class of finite trees as a subclass from the class of ordered trees of height ω which are finitely branching. To this end put

$$\pi_0 = (\mathtt{down}; \mathsf{first}?)^*; \mathsf{leaf}?,$$
$$\pi_1 = (\mathsf{last}?; \mathtt{up})^*$$

and

$$\gamma = \langle \pi_0; \pi_1; (\mathtt{right}; \pi_0; \pi_1)^* \rangle \mathsf{root}. \tag{17}$$

Before we move on let us introduce some terminology. A *branch* b in a finitely branching tree T of height ω is a sequence

$$r = x_1(\mathtt{down})x_2(\mathtt{down}) \cdots (\mathtt{down})x_n(\mathtt{down}) \cdots$$

where r is the root of T and either b is infinite and in this case is $(\mathtt{down})$ closed, or its last element is a leaf. If b and b' are branches then we say that b is *to the left* of b' whenever if i is the smallest i such that $b_i \neq b'_i$ then $b_i(\mathtt{right})^+ b'_i$. Clearly, since T is finitely branching, this gives us a linear ordering on the branches of some fixed tree.

2. In case the tree is infinitely branching the sibling order $R_{\mathtt{right}}$ might be non-total, but this does not matter, see Lemma 16.

For $t \in T$ and b a branch of T we write $t < b$ if $t \notin b$ and t occurs on some branch to the left of b. $t \leq b$ means $t < b$ or $t \in b$. Similar definitions hold for $b < t, b \leq t$.

LEMMA 17. — *Suppose T is finitely branching tree of height ω and $t \in T$. If T_t is finite then $t(\pi_0; \pi_1; (\mathtt{right}; \pi_0; \pi_1)^*)t$.*

PROOF. — Induction on height(t). Similar to the proof of Lemma 13. ■

LEMMA 18. — *Suppose T is finitely branching tree of height ω with root r. If T is infinite then not $r(\pi_0; \pi_1; (\mathtt{right}; \pi_0; \pi_1)^*)r$.*

PROOF. — Since T is infinite, finitely branching and of height ω, T must contain an infinite branch. Let b be the leftmost infinite branch of T. Such a branch can easily be constructed by starting from r and in each successive step select the leftmost child of the previously selected node which roots an infinite subtree. The following is obvious.

1) $x \leq b$ and $x(\pi_0)y$ imply $y < b$
2) $x < b$ and $x(\pi_1)y$ imply $y < b$
3) $x < b$ and $x(\mathtt{right})y$ imply $y \leq b$

1 is clear, since π_0 only walks to leftmost children. 2 is clear, since π_1 only walks from rightmost children. 3 is clear, by definition of the ordering on branches.

Now let us assume that $r(\pi_0; \pi_1; (\mathtt{right}; \pi_0; \pi_1)^*)r$. Then there exists some sequence

$$r = a_0(\pi_0)a_1(\pi_1)a_2(\mathtt{right})a_3(\pi_0)a_4 \cdots a_{k-2}(\pi_1)a_{k-1} = r.$$

By the above three points it follows, with induction on i, that

$$\text{for all } i < k, a_i \leq b. \tag{18}$$

Since $a_{k-3}(\pi_0)a_{k-2}$ we have that a_{k-2} must be a leaf, and since $a_{k-2}(\pi_1)r$ we also have that the branch in T ending in a_{k-2} only contains rightmost nodes. But this implies that b, as the leftmost infinite branch of T, must be on the left of the branch ending in a_{k-2}. So in particular $b \leq a_{k-2}$. But since a_{k-2} is a leaf we even have $b < a_{k-2}$, in contradiction with (18). ■

5.4. *The upshot*

What is the upshot of these examples? First and foremost, they were intended to show the language in action, to show that semantic reasoning is naturally captured in $\mathsf{PDL}_{\mathsf{tree}}$ formulas, even when it comes to hard problems. Even though we provided rigorous correctness proofs, we feel that once the semantic argument is understood, correctness of the $\mathsf{PDL}_{\mathsf{tree}}$ formalization is almost self-evident.

Although boolean circuits looks like a canonical MSO problem it has certain peculiarities which we could exploit, in particular that one depth-first traversal of the tree is sufficient to determine the truth of the formula. The problem suggest a possible strengthening of the language: intersection of programs with $\top?$. With this we can specify the set of all points t at which $\mathrm{eval}(t)$ is true, and not just the root.

6. Conclusions

We hope that we convinced the reader that PDL is a natural formalism for reasoning about ordered trees. We showed that it has good complexity measures, both in terms of model checking and in terms of satisfiability and consequence problems. $\mathsf{PDL}_{\mathsf{tree}}$ has natural subfragments which are expressively complete with respect to first order logic. The most pressing open problems are to determine the exact expressive power of $\mathsf{PDL}_{\mathsf{tree}}$ and to understand whether the extra expressivity given by unary MSO is useful in specific applications as linguistics or the XML-world.

Acknowledgements

Thanks are due to Michael Benedikt for suggesting the frontier and the boolean circuit problem, to Wouter Kuijpers for the formula expressing finiteness, and to Jan Hidders, Frank Neven and Hans-Joerg Tiede for valuable feedback. Part of this work was carried out as part of the INRIA funded research partnership between LIT (Language and Inference Technology Group, University of Amsterdam) and LED (Langue et Dialogue, LORIA, Nancy). Afanasiev is supported by a grant from the Netherlands Organization for Scientific Research (NWO), under project number 612.000.207 and 017.001.190. Marx and Goris are supported by a grant from NWO under project number 612.000.106, and De Rijke is supported by grants from NWO, under project numbers 365-20-005, 612.069.006, 612.000.106, 220-80-001, 612.000.207, 612.066.302, 017.001.190, and 264-70-050.

7. References

[ABI 00] ABITEBOUL S., BUNEMAN P., SUCIU D., *Data on the web*, Morgan Kaufman, 2000.

[ALE 00] ALECHINA N., IMMERMAN N., "Reachability Logic: An Efficient Fragment of Transitive Closure Logic", *Logic Journal of the IGPL*, vol. 8, num. 3, 2000, p. 325–337.

[ALE 03] ALECHINA N., DEMRI S., DE RIJKE M., "A Modal Perspective on Path Constraints", *Journal of Logic and Computation*, vol. 13, 2003, p. 1–18.

[BLA 94] BLACKBURN P., MEYER-VIOL W., "Linguistics, Logic, and Finite Trees", *Logic Journal of the IGPL*, vol. 2, 1994, p. 3–29.

[BLA 96] BLACKBURN P., MEYER-VIOL W., DE RIJKE M., "A proof system for finite trees", BÜNING H. K., Ed., *Computer Science Logic*, vol. 1092 of *LNCS*, Springer, 1996, p. 86–105.

[BLA 01] BLACKBURN P., DE RIJKE M., VENEMA Y., *Modal Logic*, Cambridge University Press, 2001.

[BLA 03] BLACKBURN P., GAIFFE B., MARX M., "Variable free reasoning on finite trees", *Proceedings of Mathematics of Language (MOL–8), Bloomington*, 2003.

[CON 04] CONSORTIUM W.-W. W., "Extensible Markup Language (XML) 1.0 (Second Edition) W3C", `http://www.w3.org/TR/REC-xml`, 2004.

[ETE 97] ETESSAMI K., VARDI M., WILKE TH., "First-order logic with two variables and unary temporal logic", *Proc. LICS'97*, 1997, p. 228–235.

[FIS 79] FISHER M., LADNER R., "Propositional dynamic logic of regular programs", *J. Comput. Syst. Sci.*, vol. 18, num. 2, 1979, p. 194–211.

[GAB 84] GABBAY D., GUENTHER F., Eds., *Handbook of Philosophical Logic*, vol. 2, Reidel, Dordrecht, 1984.

[GOT 02] GOTTLOB G., KOCH C., "Monadic Datalog and the Expressive Power of Languages for Web Information Extraction", *PODS'02*, 2002, p. 17-28.

[HAR 00] HAREL D., KOZEN D., TIURYN J., *Dynamic Logic*, MIT Press, 2000.

[KAM 68] KAMP J., "Tense Logic and the Theory of Linear Order", PhD thesis, University of California, Los Angeles, 1968.

[KRA 95] KRACHT M., "Syntactic codes and grammar refinement", *Journal of Logic, Language and Information*, vol. 4, 1995, p. 41–60.

[KRA 97] KRACHT M., "Inessential Features", RETORE C., Ed., *Logical Aspects of Computational Linguistics*, num. 1328 LNAI, p. 43–62, Springer, 1997.

[KRA 99] KRACHT M., *Tools and Techniques in Modal Logic*, Num. 142 Studies in Logic, Elsevier, Amsterdam, 1999.

[MAR 04a] MARX M., "Conditional XPath, the first order complete XPath dialect", *Proceedings of PODS'04*, 2004, p. 13–22.

[MAR 04b] MARX M., DE RIJKE M., "Semantic Characterizations of XPath", *TDM'04 workshop on XML Databases and Information Retrieval.*, Twente, The Netherlands, June 21, 2004.

[MAR 05] MARX M., "First order paths in ordered trees", EITER T., LIBKIN L., Eds., *Proc. ICDT 2005*, vol. 3363 of *LNCS*, 2005, p. 114-128.

[NEV 00a] NEVEN F., SCHWENTICK T., "Expressive and Efficient Pattern Languages for Tree-Structured Data", *Proc. PODS*, ACM, 2000, p. 145–156.

[NEV 00b] NEVEN F., SCHWENTICK T., "On the Power of Tree-Walking Automata", U. MONTANARI J. ROLIM E. W., Ed., *Automata, Languages and Programming, Proceedings, ICALP 2000, LNCS 1853*, Springer, 2000, p. 547-560.

[PAL 97] PALM A., "Transforming tree constraints into formal grammars", PhD thesis, Universität Passau, 1997.

[PAL 99] PALM A., "Propositional tense logic for trees", *Sixth Meeting on Mathematics of Language*, University of Central Florida, Orlando, Florida, 1999.

[PRA 79] PRATT V., "Models of Program Logics", *Proceedings of the 20th IEEE symposium on Foundations of Computer Science*, 1979, p. 115-122.

[ROG 98] ROGERS J., *A Descriptive Approach to Language Theoretic Complexity*, CSLI Press, 1998.

[SPA 93] SPAAN E., "Complexity of modal logics", PhD thesis, University of Amsterdam, Institute for Logic, Language and Computation, 1993.

[VAR 86] VARDI M., WOLPER P., "Automata-theoretic techniques for modal logics of programs", *Journal of Computer and System Sciences*, vol. 32, 1986, p. 183–221.

[WEY 02] WEYER M., "Decidability of S1S and S2S", ET AL. E. G., Ed., *Automata, Logics, and Infinite Games*, vol. 2500 of *LNCS*, p. 207–230, Springer, 2002.